

design ideas

Edited by Bill Travis and Anne Watson Swager

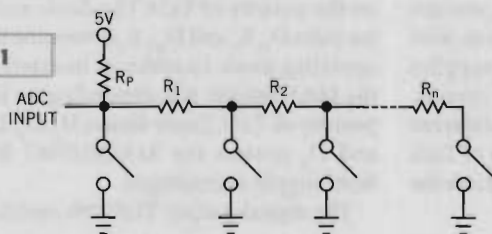
ADC circuit optimizes key encoding

Vitor Amorim, Siemens, and J Simões, University of Coimbra, Coimbra, Portugal

YOU CAN IMPLEMENT a simple and low-cost keyboard encoder by using a key-controlled resistive divider connected to an ADC (Figure 1). This type of circuit is especially suitable for embedded systems that use a μ C with an integrated ADC section. To obtain the best noise margin between keys, select resistors to yield an equal division of the voltage levels. To meet this criterion using the circuit in Figure 1, you must use a set of resistors with all-different, specific values. For example, a 10-key keyboard uses a 10-k Ω pullup resistor, R_p , and values of 1.1, 1.3, 1.8, 2.4, 3.3, 5.1, 8.2, 16, and 51 k Ω for R_1 through R_9 . Using commonly available resistor values and tolerances and taking account of the key resistivity and ADC linearity errors, you're limited in the number of keys you can encode with a safe noise margin. Using an 8-bit ADC, the cited 10-key example is close to that limit.

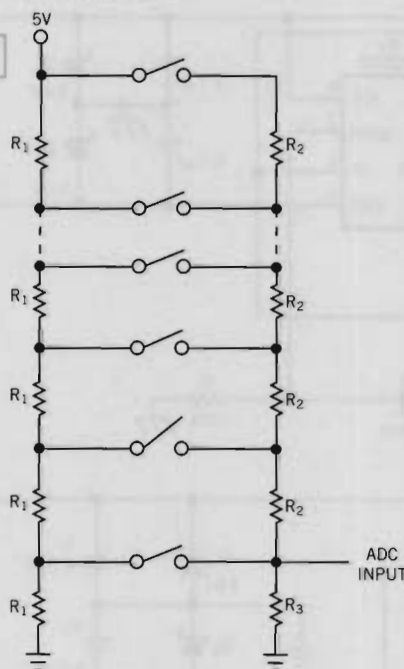
The circuit in Figure 2, despite its many resistors, overcomes the problem of the many restore values. The circuit is symmetric and uses the same two resis-

Figure 1



This is a simple way to encode keys, but it requires a range of resistor values for a large number of keys.

Figure 2



This key-encoding circuit simplifies resistor inventories, and provides a comfortable noise margin to boot.

tor values for all keys. It is also easy to expand the circuit for more keys. As in Figure 1's circuit, if you simultaneously press more than one key, the circuit detects only the key whose connection is closest to the ADC. The chain of R_i resistors de-

fines the voltage level associated with each key. Their nominal value is a trade-off between the power dissipation and the values of R_2 and R_3 , which depend on R_1 ; the total number of keys; and the desired noise margin. R_2 minimizes the voltage deviations at the nodes of the R_i chain whenever you simultaneously press two or more keys. Therefore, you should calculate its value, much higher than that of R_1 , by taking into account R_1 and the desired width of the voltage window associated with each key. Similarly, R_3 's value should be much higher than that of R_2 to ensure that the voltage level associated with each key almost completely transfers to the ADC's input.

The circuit was tested by encoding 15 keys with one 8-bit analog input of the Microchip PIC16C71 μ C. The resistor values are 47 Ω , 3.9 k Ω , and 4.7 M Ω for R_1 , R_2 , and R_3 , respectively. Only R_1 needs a tight tolerance; the others are less demanding. The window of key acceptance is set to a 7-LSB interval centered on the theoretical key level. This interval is large enough to accommodate the worst case (simultaneously pressing the two more-distant keys from the analog input) with a safety noise margin between valid keys of 10 LSBs. (DI #2319).

TO VOTE FOR THIS DESIGN,
CIRCLE NO. 332

ADC circuit optimizes key encoding	101
RS-232C circuit has galvanic isolation	102
Digital pot adjusts LCD's contrast	104
Add speech encoding/decoding to your design	106
Cheap PWM IC makes synchronous gate driver	110
Circuit detects total on-time	112

RS-232C circuit has galvanic isolation

Ioan Ciascai, REI Data, Napoca, Romania

YOU CAN OBTAIN LONGER TRANSMISSION distances with the RS-232C interface if you use galvanic isolation between the two linked terminals. Galvanic isolation also eliminates problems arising from disparate potentials between terminals. Using two MAX860/861 ICs and two TPL2200 optoisolators, you can obtain galvanic isolation for three-wire transmission without external supplies (Figure 1). The MAX860/861 circuits, which generate two voltages of different polarity, regardless of the polarity of TxD, which provides the supply voltage, are the basis of the design.

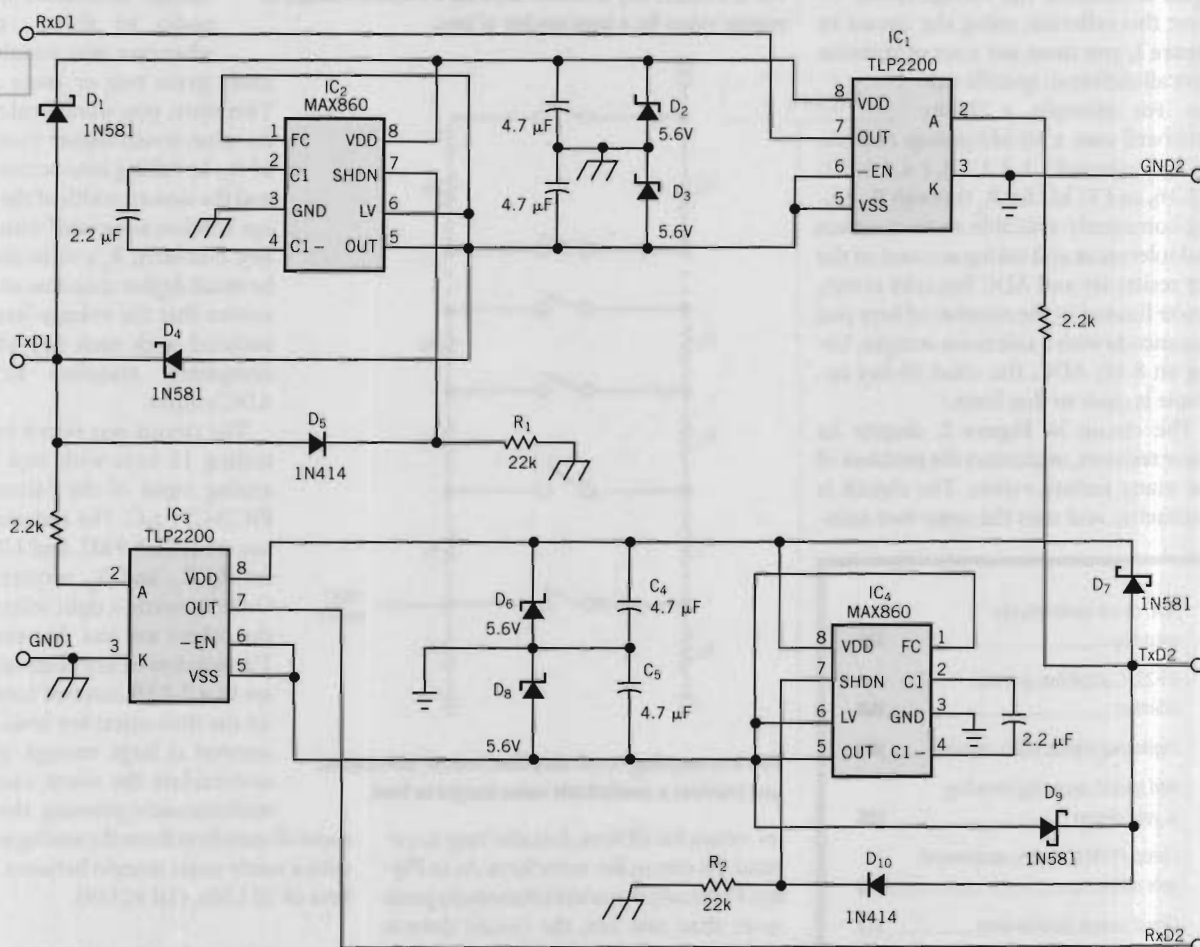
For the positive TxD polarity, the MAX860/861 ICs function as voltage inverters, whereas for negative-polarity TxD, the ICs function as voltage doublers. Diodes D₁ and D₇ provide the positive supply voltages; D₄ and D₉ provide the negative supply voltages, depending on the polarity of TxD. The diode-resistor pairs D₅, R₁ and D₁₀, R₂ determine the operating mode (doubler or inverter) of the MAX860/861 ICs, depending on the polarity of TxD. Zener diodes D₂, D₃, D₆, and D₈ protect the MAX860/861 ICs from supply overvoltages.

The digital-output TPL2200 optoisolators

provide galvanic isolation and generate the RxD received signals with RS-232C logic levels. The circuit provides galvanically isolated communication in full-duplex mode at any standard transmission speed. The use of galvanic isolation allows transmission over nearly twice the distance for nonisolated systems. If you use a four-wire cable and split the separation circuits at the cable ends, you can further increase the transmission distance. (DI #2315).

TO VOTE FOR THIS DESIGN,
CIRCLE NO. 333

Figure 1



RS-232C-interface ICs and optoisolators provide a supplyless RS-232C transmission link with galvanic isolation for increased distance.

Digital pot adjusts LCD's contrast

Jef Collin, CSE Systems, Turnhout, Belgium

YOU CAN USE A DIGITALLY controlled potentiometer for many purposes. In this example, you can use the device to regulate the contrast of a standard (such as two lines by 40 characters) LCD. You can use the circuit in **Figure 1** in a portable test system, in which you need to change the contrast of the LCD as a function of the viewing angle. You choose the contrast setup from a menu and then use up or down buttons with μC IC₁ to adjust the contrast. The μC stores the contrast value in the digital potentiometer, IC₂. This design uses a Xicor 10-k Ω unit (dubbed "EEPOT"), but you could use other devices in the design.

The EEPOT connects to the LCD's VO line. You could connect the other side of the potentiometer to ground, but, for better contrast, you can apply a negative voltage to this terminal. This circuit has a

LISTING 1—ROUTINE FOR LCD-CONTRAST ADJUSTMENT

```

; *** BIT DEFINITIONS ***
EEPOT_INC BIT P1.4      ; EEPOT CONNECTIONS
EEPOT_UD BIT P1.5
EEPOT_CS BIT P1.6

; *** INITIAL SETTINGS ***
SETB EEPOT_CS
SETB EEPOT_INC
SETB EEPOT_UD

; *** UP CONTRAST ***
CONT_UP:  NOP
          SETB EEPOT_UD
          SETB EEPOT_INC
          NOP
          CLR EEPOT_CS
          NOP
          CLR EEPOT_INC
          NOP
          SETB EEPOT_INC
          NOP
          CLR EEPOT_INC
          NOP
          SETB EEPOT_INC
          NOP
          CLR EEPOT_INC
          NOP
          SETB EEPOT_CS
          CLR EEPOT_INC

          NOP
          SETB EEPOT_INC
          NOP
          CLR EEPOT_UD
          SETB EEPOT_INC
          NOP
          CLR EEPOT_CS
          NOP
          CLR EEPOT_INC
          NOP
          SETB EEPOT_INC
          NOP
          CLR EEPOT_INC
          NOP
          SETB EEPOT_CS
          CLR EEPOT_INC
          RET

; *** DOWN CONTRAST ***
CONT_DWN: NOP
          CLR EEPOT_UD
          SETB EEPOT_INC
          NOP
          CLR EEPOT_CS
          NOP
          CLR EEPOT_INC
          NOP
          SETB EEPOT_INC
          NOP
          CLR EEPOT_INC
          NOP
          CLR EEPOT_INC
          NOP
          SETB EEPOT_INC
          NOP
          CLR EEPOT_INC
          NOP
          SETB EEPOT_CS
          CLR EEPOT_INC
          RET

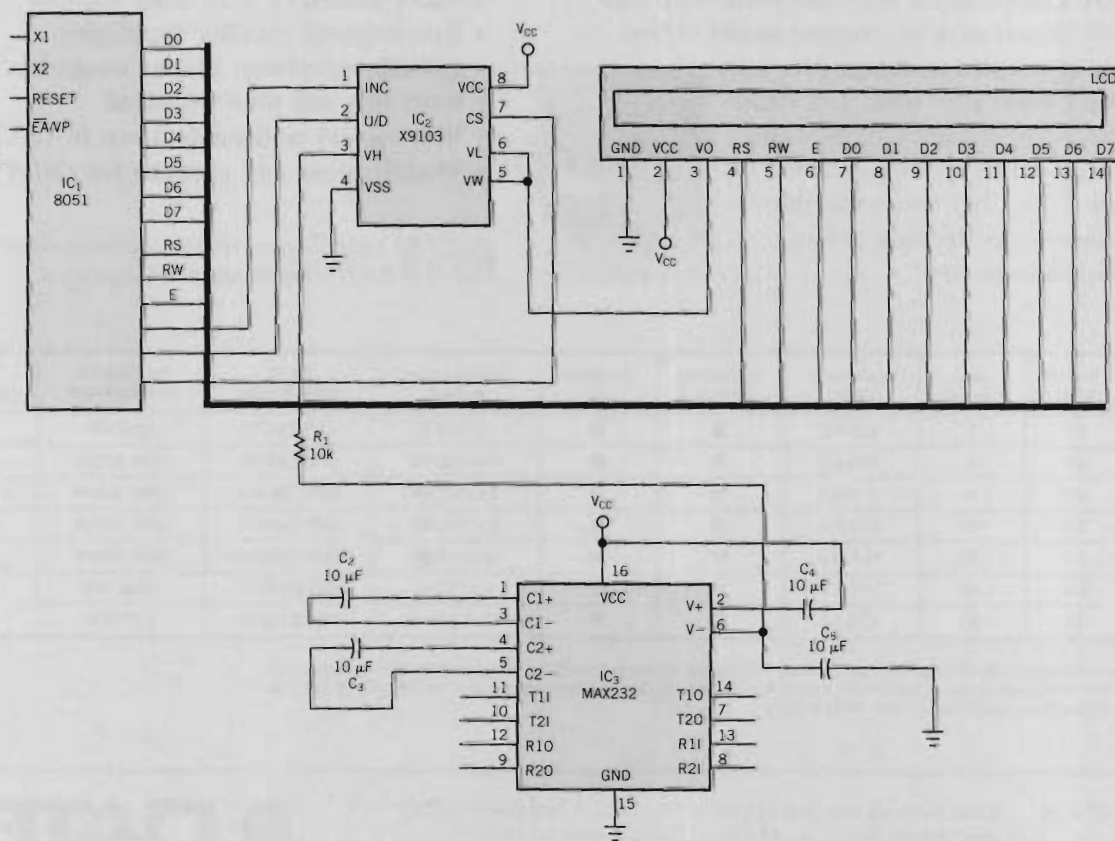
```

serial interface, so it uses the negative-voltage generator in the MAX232 RS-232C/TTL converter. **Listing 1** is the subroutine for the 8051 μC . The program

calls the routines for contrast-up or -down. (DI #2314).

TO VOTE FOR THIS DESIGN,
CIRCLE No. 334

Figure 1



A standard 8051 μC and a digitally controlled potentiometer provide a convenient way to vary the contrast of an LCD, using contrast-up and -down buttons.

Add speech encoding/decoding to your design

Rodger Richey, Microchip Technology Inc, Chandler, AZ

ADDING SPEECH capabilities to a design can sometimes lead to complex algorithms and expensive DSPs or specialized audio chips. However, with the completion of a simplified adaptive differential pulse-code-modulation (ADPCM) algorithm, you can now implement these audio capabilities in low-cost 8-bit μ Cs, which typically have lower power consumption and cost than their DSP or audio-chip counterparts. A two-chip design is feasible by offloading the encoding and decoding tasks onto the μ C as if it were a peripheral.

Since 1991, the Interactive Multimedia Association (IMA) Digital Audio Technical Working Group (DATWG) has been working to define a cross-platform digital-audio exchange format. An inherent problem exists with the exchange of audio data between PC, Mac, and workstation computers. Each computer has its own data types and sampling rates. In May 1992, IMA DATWG published the first revision of the Cross-Platform Digital Audio Interchange recommendation that specifies three uncompressed and one compressed data type at various sample rates. The compressed data type is the Intel (www.intel.com) 4-bit DVI ADPCM algorithm. This algorithm compresses a 16-bit signed audio sample into 4 bits and takes advantage of the high correlation between consecutive samples, which enables the prediction of future samples. Instead of encoding the sample itself, ADPCM encodes the difference between a predicted sample and the actual sample. This method provides more efficient

compression with fewer bits per sample and yet preserves the overall quality of the audio signal. Both the encoder routine (Listing 1) and the decoder routine (Listing 2) are written in C to ease readability.

LISTING 1—ADPCM ENCODER

```
/* Table of index changes */
const char IndexTable[16] = {-1,-1,-1,-1,2,4,6,8,-1,-1,-1,-1,2,4,6,8};

/* Quantizer step size lookup table */
const long StepSizeTable[89] = {7,8,9,10,11,12,13,14,16,17,19,21,23,25,28,31,34,37,41,
45,50,55,60,66,73,80,88,97,107,118,130,143,157,173,190,
209,230,253,279,307,337,371,408,449,494,544,598,658,724,
796,876,963,1060,1166,1282,1411,1552,1707,1878,2066,2272,
2499,2749,3024,3327,3660,4026,4428,4871,5358,5894,6484,
7132,7845,8630,9493,10442,11487,12635,13899,15289,16818,
18500,20350,22385,24623,27086,29794,32767};

signed int diff;          // Diff. between sample and predicted sample
int step;                // Quantizer step size
signed int predsamp, diffq; // ADPCM predictor output, Predicted diff.
char index;              // Index into step size table

char ADPCMEncoder( signed int sample )
{
    char code;            // ADPCM output value

    predsamp = state.prevsamp; // Restore previous values of predicted
    index = state.previndex;  // sample and quantizer step size index
    step = StepSizeTable[index];

    diff = sample - predsamp; // Compute diff. between actual sample
    if( diff >= 0 )           // and the predicted sample
        code = 0;
    else
    {
        code = 8;
        diff = -diff;
    }

    diffq = step >> 3;        // Quantize the diff. into 4-bit ADPCM code
    if( diff >= step )        // using the quantizer step size
    {
        code |= 4;           // Inverse quantize the ADPCM code into a
        diff -= step;         // predicted diff. using the quantizer step
        diffq += step;        // size
    }
    step >>= 1;
    if( diff >= step )
    {
        code |= 2;
        diff -= step;
        diffq += step;
    }
    step >>= 1;
    if( diff >= step )
    {
        code |= 1;
        diffq += step;
    }

    if( code & 8 )            // Compute new predicted sample by adding
        predsamp += diffq;   // the old predicted sample to new diff.
    else
        predsamp -= diffq;
    if( predsamp > 32767 )    // Check if overflow of the new pred. sample
        predsamp = 32767;
    else if( predsamp < -32768 )
        predsamp = -32768;

    index += IndexTable[code]; // Find new quantizer stepsize
    if( index < 0 )           // Check if overflow of new quantizer step
        index = 0;
    if( index > 88 )
        index = 88;

    state.prevsamp = predsamp; // Save values for next iteration
    state.previndex = index;
    return ( code & 0x0f );    // Return the ADPCM code
}
```

The hardware implementation depends on the type of interface. With a parallel interface, you can use the PIC16C556A (Microchip Technology, www.microchip.com) (Figure 1a). A standard parallel interface uses the chip-